COSC 85

COSC 85

LECTURE 3 - NETWORK PROTOCOLS

NETWORK PROTOCOLS

- are standardized rules and procedures that govern how devices communicate and exchange data over a network, acting as a common language for different hardware and software to understand each other, defining data format, addressing, error handling, and transmission.
- They break down complex tasks into smaller functions, ensuring smooth, reliable, and secure communication for everything from web browsing (HTTP) to email (SMTP) and file transfers (FTP).

What Network Protocols Do?

- **Enable Communication:** Allow diverse devices (PCs, servers, phones) to talk, regardless of internal differences.
- **Define Data Handling:** Specify data formatting, addressing, packaging, and error checking.
- **Manage Flow:** Act like traffic signals to regulate data movement, preventing errors and ensuring delivery.

KEY EXAMPLES (TCP/IP SUITE)

1. **HTTP (Hypertext Transfer Protocol):** HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol.
2. **HTTP/3:** HTTP/3 is the next major revision of the HTTP. It runs on QUIC, a new transport protocol designed for mobile-heavy internet usage. It relies on UDP instead of TCP, which enables faster web page responsiveness.
3. **HTTPS (HyperText Transfer Protocol Secure):** HTTPS extends HTTP and uses encryption for secure communications.
4. **WebSocket:** WebSocket is a protocol that provides full-duplex communications over TCP. Clients establish WebSockets to receive realtime updates from the back-end services. Unlike REST, which always “pulls” data, WebSocket enables data to be “pushed”.
5. **TCP (Transmission Control Protocol):** TCP is designed to send packets across the internet and ensure the successful delivery of data and messages over networks. Many application-layer protocols build on top of TCP.
6. **UDP (User Datagram Protocol):** UDP sends packets directly to a target computer, without establishing a connection first. UDP is commonly used in time-sensitive communications where occasionally dropping packets is better than waiting. Voice and video traffic are often sent using this protocol.
7. **SMTP (Simple Mail Transfer Protocol):** SMTP is a standard protocol to transfer electronic mail from one user to another.
8. **FTP (File Transfer Protocol):** FTP is used to transfer computer files between client and server. It has separate connections for the control channel and data channel.

LECTURE 4 - CISCO PACKET TRACER

1. **Menu Bar** - Contains standard file, edit, options, view, tools, and help menus, providing access to basic commands like open, save, and print.
2. **Main Tool Bar** - Offers shortcut icons for frequently used menu commands, including new, open, save, print, activity wizard, copy, paste, undo, and redo.
3. **Logical/Physical Workspace Tabs** - These tabs allow you to toggle between the Logical and Physical work areas.
4. **Common Tools Bar** - Offers access to primary workspace tools, including Select, Inspect, Delete, Resize Shape, Place Note, Drawing Palette, Add Simple PDU (packet icon), and Add Complex PDU.
5. **Workspace** - This is the area where topologies are created and simulations are displayed.
6. **Realtime/Simulation Tabs** - These tabs are used to toggle between the real and simulation modes. Buttons are also provided to control the time, and to capture the packets.
7. **Network Component Box** - This component contains all of the network and

end devices available with Packet Tracer, and is further divided into two areas:

    a. **Network Component Box (Device-type selection box)** - This area contains device categories.
    b. **Network Component Box (Device-specific selection box)** - When a device category is selected, this selection box displays the different device models within that category.

8. **User-created Packet Box** - Users can create highlycustomized packets to test their topology from this area, and the results are displayed as a list.

User Levels and Modes

1. Router > - User EXEC Mode. This mode is primarily for read-only monitoring. Users can check basic hardware and software status using limited show commands (e.g., show version, show history). You can also perform basic connectivity tests like ping and traceroute.

2. Router # - Privileged EXEC Mode. Often called "Enable Mode," this level grants full administrative visibility. It is the "hub" for advanced troubleshooting and management.

3. Router(Config) # - Global Configuration Mode. This is where you implement system-wide changes that affect the entire device.

4. Router(Config-if) # - Interface Mode. Used to configure individual physical or virtual ports. Here you assign IP addresses, set link speeds, or enable/disable ports with no shutdown.

5. Router(Config Router) # - Routing Engine Mode. Used for setting up routing protocols. Inside this mode, you define which networks a protocol like OSPF or RIP should advertise to other devices.

6. Router(Config-Line) # - Line Mode. Used to secure access points. You enter this mode to set passwords for the console (physical access) or vty (remote Telnet/SSH access) lines.

Other Cisco Packet Necessity

- **Running Configuration** - The running configuration, also known as the "current configuration" or "active configuration" is the configuration that is currently active and operational on the network device. Changes made to the running configuration take effect immediately and impact the device's operation as soon as they are applied using commands like 'configure terminal' or by using a configuration management tool. These changes are not saved across device reboots.
- **Startup Configuration** - The startup configuration, also known as the "saved configuration" or "boot configuration" is the configuration that is saved on the device's non-volatile memory. It represents the configuration that will be loaded and applied to the device when it boots up or is restarted. Changes made to the startup configuration are saved across reboots, ensuring that the device maintains the desired configuration even after power cycles or unexpected reboots.
- **Configuration Register** - is a 16-bit software value stored in a Cisco device's non-volatile RAM (NVRAM) that acts as a set of boot instructions.

Configuration Register (0x2142) – Password Recovery Mode

- This setting is specifically used when you need to bypass existing security settings.
- **Behavior:** The router loads the IOS from Flash but completely ignores the startup-config in NVRAM.
- **Purpose:** Since the router ignores the startup-config, it doesn't prompt for a password when you enter enable mode. Once inside, you can manually load the old configuration, change the forgotten password, and then save the new settings.

Configuration Register (0x2102) – Default Mode

- This is the factory-default setting for most Cisco routers.
- **Behavior:** The router boots normally by loading the Cisco IOS software from Flash memory and the startup-config from NVRAM.
- **Purpose:** Used for standard day-to-day operations where the router is expected to load its saved settings (passwords, IP

addresses, etc.) immediately upon power-up.

PASSWORD ENCRYPTION

- **Global Password Encryption** - The service password-encryption command is used to encrypt all current and future plaintext passwords (such as console and VTY passwords) in the configuration file.
- **How it works:** It uses Cisco Type 7 encryption, which is a weak, reversible algorithm designed only to prevent "shoulder surfing" (others seeing passwords while you view the configuration).
- **Command:** Router(config)# service password-encryption.
- **Privileged EXEC Mode Encryption** - There are two commands to set a password for entering privileged mode, each offering different levels of security:
  - ➢ **enable secret <password>:** This is the recommended method. It uses the MD5 algorithm (Type 5) to hash the password, making it much more secure and difficult to reverse than Type 7.
  - ➢ **enable password <password>:** This stores the password in plaintext by default. It only becomes encrypted (as Type 7) if the service password encryption command is also active.

Why Password Encryption is important?

- **Mitigation of Data Breaches** - When a database is compromised, encryption ensures that attackers find only unreadable "ciphertext" rather than usable plaintext credentials. You should use modern hashing algorithms to ensure that even if data is stolen, the passwords remain computationally impossible to crack for years.
- **Prevention of Credential Stuffing** - Many users reuse the same password across multiple sites, meaning a single plaintext leak can grant attackers access to a user's entire digital life. You must implement robust encryption so that a compromise on your system does not become a gateway for attackers to hijack accounts on other platforms.
- **Protection Against Future Threats** - Adversaries are increasingly collecting encrypted data with the intent to decrypt it once quantum computing becomes more commercially viable. You should proactively migrate to post-quantum cryptography or hybrid encryption models to protect today's sensitive data against these future decryption capabilities.
- **Regulatory and Legal Compliance** - Modern data protection laws strictly require the encryption of sensitive personal data to protect consumer privacy and digital rights. You can avoid massive regulatory fines and devastating lawsuits by maintaining an encrypted environment that meets current international security standards.
- **Defending Against Insider Threats** - Encryption prevents unauthorized internal staff, such as system administrators, from viewing user passwords within the configuration files or databases. You should apply the principle of least privilege alongside encryption to ensure that no single person has the ability to view sensitive credentials in plaintext.
- **Maintaining Brand Trust and Reputation** - A breach involving plaintext passwords causes immediate and often permanent erosion of customer trust, which directly impacts a company's market value. You can build long-term stakeholder confidence by being transparent about your encryption protocols and demonstrating a security-first approach to user data.

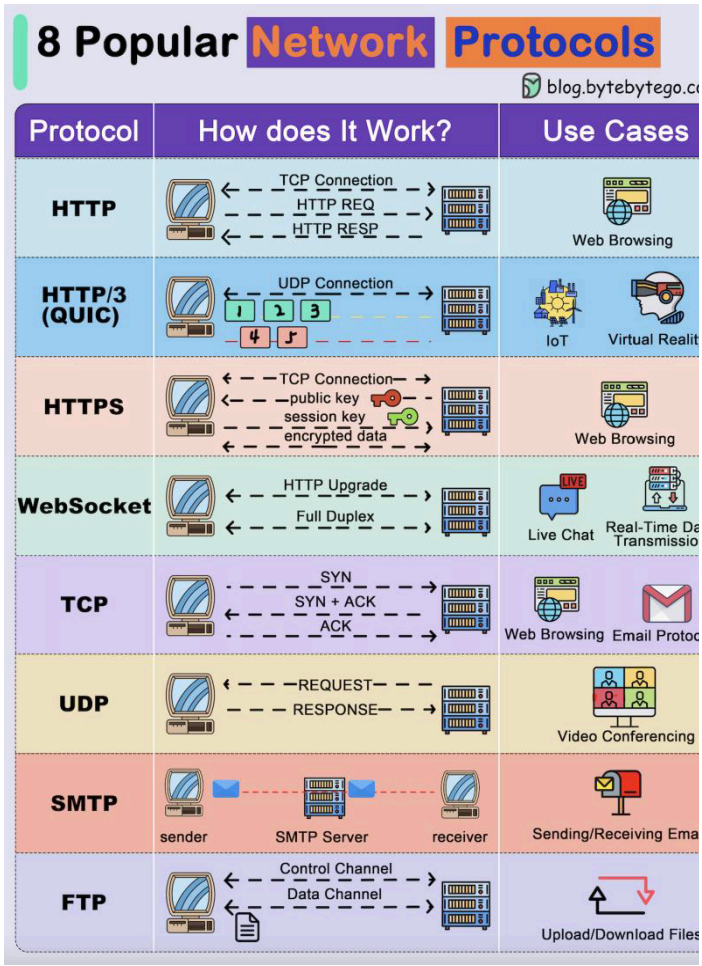
LECTURE 5 - NETWORK TROUBLESHOOTING

Common Network Issues:

1. **Slow Network** - High bandwidth usage from activities like 4K streaming or large file downloads saturates the connection, often exacerbated by outdated hardware that cannot handle modern internet speeds.
   a. **Possible Solution:** You should implement Quality of Service

(QoS) settings to prioritize essential traffic or perform a speed test to ensure your ISP is delivering the bandwidth promised in your service agreement.

2. **Weak Wi-Fi Signal** - Signal strength degrades due to physical distance from the router or interference from thick walls, metal objects, and other electronic devices operating on the same frequency.
    a. **Possible Solution:** You can resolve this by moving your router to an elevated, central location or by deploying a mesh Wi-Fi system to provide consistent coverage throughout the building.
3. **Physical Connectivity Issues** - Sudden outages or intermittent connections are frequently caused by loose connectors, damaged Ethernet cables, or faulty hardware ports.
    a. Possible Solution: You must inspect all physical lines for kinks or damage and ensure that every cable is firmly seated and "clicked" into its respective port.
4. **Excessive CPU Usage** - Routers and switches may become sluggish or unresponsive when their processors are overwhelmed by high traffic volumes, malware, or complex encryption tasks.
    a. **Possible Solution:** The most effective fix is to power cycle the device to refresh the memory and then disable any unnecessary background services or outdated protocols.
5. **Slow DNS Lookups** - When an ISP's DNS server is slow or non-responsive, the time it takes to translate a website name into an IP address increases, leading to long loading delays.
    a. **Possible Solution:** You should configure your router or computer to use high-performance DNS providers like Cloudflare (1.1.1.1) or Google (8.8.8.8) to ensure nearinstant resolution.
6. **Duplicate and Static IP Addresses** - Connectivity fails for one or more devices when two pieces of hardware are assigned the same IP address, causing a conflict on the network.
    a. **Possible Solution:** You can resolve this conflict by ensuring all devices are set to obtain addresses automatically via DHCP or by carefully auditing static IP assignments to ensure they are unique.
7. **Exhausted IP Address** - This occurs when the DHCP server has no more available addresses to hand out, preventing new devices from joining the network.
    a. **Possible Solution:** You should expand the range of available IP addresses in your router's DHCP settings or shorten the lease duration to reclaim addresses from disconnected devices faster.
8. **Virtual LAN and VPN problems** - These issues typically involve incorrect VLAN tagging on network switches or authentication and encryption failures within a VPN tunnel.
    a. **Possible Solution:** You must verify that trunk ports are correctly configured for VLAN traffic and ensure that VPN clients are updated to the latest 2026 security standards to prevent connection drops.


8 Popular Network Protocols
blog.bytebytego.c
Protocol
How does It Work?
Use Cases
HTTP
TCP Connection
HTTP REQ
HTTP RESP
Web Browsing
HTTP/3 (QUIC)
UDP Connection
IoT
Virtual Realit
HTTPS
TCP Connection
public key
session key
encrypted data
Web Browsing
WebSocket
HTTP Upgrade
Full Duplex
Live Chat
Real-Time Da Transmissio
TCP
SYN
SYN + ACK
ACK
Web Browsing
Email Protoc
UDP
REQUEST
RESPONSE
Video Conferencing
SMTP
sender
SMTP Server
receiver
Sending/Receiving Ema
FTP
Control Channel
Data Channel
Upload/Download Files

Alesya

COSC 101

COSC 101
LECTURE 4 - INTERACTIVE COMPUTER GRAPHICS

What is GAME DESIGN?

- Multidisciplinary field combining computer science, creative writing, and graphic design.
- Planning phase of game creation (serves as a blueprint)
- Requires creativity and technical skills such as logic, game theory, and knowledge of human psychology.

GAME DEVELOPMENT SOFTWARES

1. **GameMaker Studio:** beginner friendly with drag-drop interface.
2. **Unity:** Powerful industry-standard for creating 2D and 3D games for numerous platforms.
3. **Unreal Engine:** A professional, complete suite of tools for high-end, real-time 3D creation.
4. **Construct 3:** Code-free, entirely GUI-driven development.
5. **Godot:** Free engine with strong, well-designed support for both 2D and 3D games.
6. **Defold:** Free software supporting export to a wide range of platforms.
7. **RPG Maker M Z:** Specialized, code-free tool for creating role-playing games.

What is LEVEL DESIGN?

- The discipline of creating game levels, missions, or environments using level editor software. A technical and artistic process that can involve conceptual sketches and models.

STEPS IN LEVEL DESIGN

1. Laying out the map (cities, rooms, tunnels).
2. Setting environmental conditions (time, weather).
3. Defining game rules (scoring, resources).
4. Placing interactive elements (buttons, teleporters, resources).
5. Adding entities (enemies, items, save points).
6. Detailing with textures, sound, lighting, and music.
7. Scripting events and creating NPC paths and behaviors.

THE 3C'S

The 3C's define the player's core interactive experience and initial impression of the game.

- **Character:** The player's avatar. Design (including silhouette) communicates personality and information (e.g., health). Customization allows for player expression.
- **Camera:** The player's "eyes" into the game world. Crucial in 3D games, with different perspectives (e.g., first-person, third-person, isometric) serving different purposes.
- **Controls:** The interface between the player and the game. Should be accessible and intuitive, unless unconventional controls are a deliberate design choice.

GAME TYPES AND DIFFICULTIES

- **Simulators:** Interactive systems that simulate something.
- **Games vs. Contests:** All games are contests, but not all contests are games. Games require meaningful decisions.
- **Puzzles:** Problems with a single, correct solution.
- **Dynamic Difficulty Adjustment (DDA):** A method to automatically modify a game's challenge in real-time based on the player's skill to avoid boredom or frustration.

CREATING GAME ASSETS

- Assets include graphics, music, and sound effects.
- ***Graphics Tools:*** Blender (3D suite), Maya (3D animation/modeling), Gimp (image editing), Aseprite/Pyxel Edit/GraphicsGale (pixel art & animation), Open Game Art (free asset library).
- ***Font Resources:*** Dafont, Google Fonts.

Music and Sound Effects

- ***Tools & Resources:*** Audiokinetic Wwise (professional audio engine), Freesound (CC-licensed sound database), BeepBox (online chiptune tool), Bosca Ceoil

(beginner-friendly music creator), Sunvox (modular music tracker).

TYPES OF 3D ANIMATION

1. ***3D Video and Film (Passive):*** Moving images in a 3D space with no user interaction.
2. ***Interactive 3D:*** Users can interact with and move through the 3D world via mouse/keyboard.
3. ***Virtual Reality 3D (VR):*** The most immersive type, requiring a headset to interact with a digital realm in a physical space.

3D ANIMATION TECHNIQUES

- **Inverse Kinematics:** Mimics natural joint movement (e.g., for arms/legs), simplifying animation.
- **Fluid Simulation:** Creates realistic animations for water, lava, etc.
- **3D Skeletal Animation:** Animates a character using a rig/skeleton underneath a skin/mesh.

What is PLAYTESTING?

- A quality control method where users play unfinished versions to find flaws and bugs. It helps refine gameplay, balance, and fun elements.

Four-Stage Process

1. **Gross Playtesting:** Initial test by the core team for basic gameplay problems.
2. **In-House Playtesting:** More comprehensive testing by internal staff/contractors.
3. **Blind Testing:** Beta testing by new users to provide fresh, unbiased feedback.
4. **Final Playtesting:** Last test before launch, focusing on aesthetic feedback.

Finalizing Your Game

- Involves making final pitching decisions and marking games as complete ("finalizing"). Finalized games can be reviewed and, if necessary, re-opened for corrections.

LECTURE 5 - 3D TRANSLATION AND TRANSFORMATION

3D TRANSFORMATION

- **Transformation** – refers to the modification or changing of position of an object.
- **3D transformation** – refers to the modification of the 3D object from its original position by modifying the physical attribute (length, width, etc.) using various transformation methods such as scaling, rotating, transforming, etc. (Nagiging iba ang posisyon ng isang 3D object kapag binago mo ang itsura nito, for example, kapag pinalaki mo yung 3D object, iba na ang position niya kahit hindi mo nilipat.)

PROPERTIES OF 3D TRANSFORMATION

1. Lines are preserved.
   (The lines remain the same even after the object was scaled, rotate, moved, etc.).
2. Parallelism is preserved.
   (The lines parallel to each other remains the same even after the transformation such as scaling, rotating, etc.).
3. Proportional distance is preserved.
   (The distance between the points remain the same after the transformation).

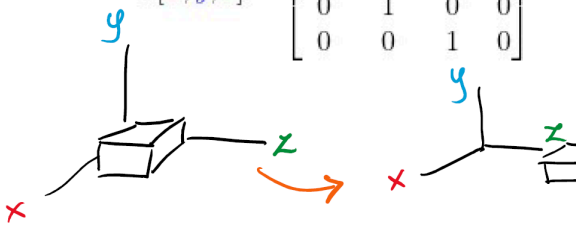
3D object's representation can be considered whether it was the surface of the object is represented or the volume of the object is represented.

1. **Boundary-based:** (Also called as b-rep)
   - It represents the surface wherein the polygon meshes, implicit surfaces, parametric surface are commonly represented by boundary-based 3D representation.
2. **Volume-based:**
   - It represents the volume of the object where in the Voxels and Constructive Solid Geometry (CSG) are commonly used to represent volumetric data.

TYPES OF TRANSFORMATION

1. **TRANSLATION**- Used for changing the position of the 3D object from its original position by coordinates where the Dx, Dy, and Dz are called Translation Distances.

$$T[x,y,z] = \begin{bmatrix} Dx & Dy & Dz & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



2. **ROTATION**
   a. If 3D, the rotation deals with x, y, and z for that defines of changing the of the 3D object. When the 3D object rotates, the specific axis is unchanged and the remaining changed. There are three kinds of rotation:
      i. Rotation about the x-axis X-axis is unchanged and the other axis changed.
      ii. Rotation about the y-axis Y-axis is unchanged and other axis changed.
      iii. Rotation about the z-axis Z-axis is unchanged and other axis changed.
   b. If 2D, it is not the same rotation to 3D.

3. **SCALING** - Used for changing the size by altering the x, y, and z direction through Sx, Sy, and Sz that is called Scaling Factors.

$$S[x,y,z] = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. **SHEARING**- Used to slant the object in 3D plane in either x, y, or z direction (this does not change the shape of the object.). It also has three types of shearing transformation:

   a. **Shearing in x-direction** - Coordinate of x-direction, unchanged. While other coordinates direction, changed. This can be defined by transformation matrix.

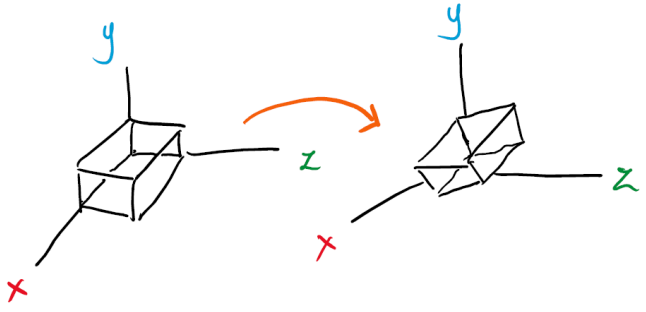
$$S[x,y,z] = \begin{bmatrix} 1 & Sy & Sz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S[x,y,z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sx & 1 & Sz & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   b. **Shearing in y-direction** - Coordinate of y-direction, unchanged. While other coordinates direction, changed. This can be defined by transformation matrix.

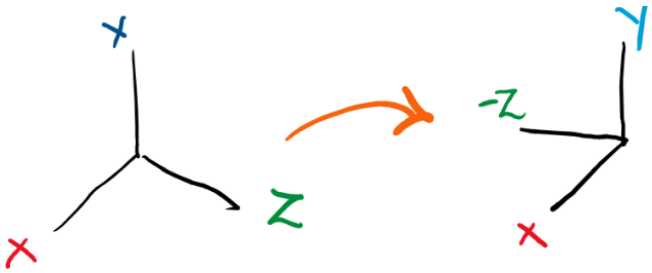
   c. Shearing in z-direction - Coordinate of z-direction, unchanged. While other coordinates direction, changed. This can be defined by transformation matrix.

$$S[x,y,z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Sx & Sy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

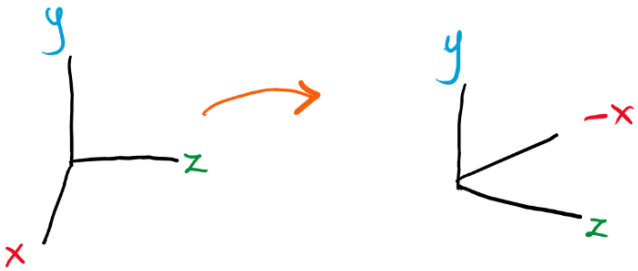


Alesya

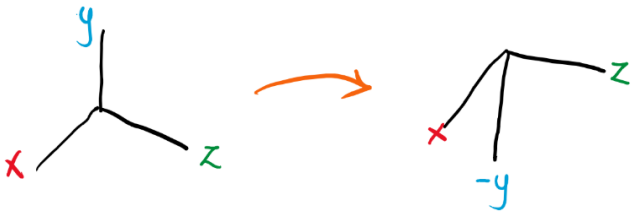
5. **REFLECTION** - It mirrors an object and has three types of reflection:
    a. **Reflection along the X-Y plane.** X-direction and Y-direction are unchanged, and the Z is changed opposite by using negative number.



    b. **Reflection along the Y-Z plane.** Y-direction and Z-direction are unchanged, and the X is changed opposite by using negative number.



    c. **Reflection along the X-Z plane.** X-direction and Z-direction are unchanged, and the Y is changed opposite by using negative number.



LECTURE 6 - BASIC LIGHTING AND ENVIRONMENT

LIGHTING SCENE

- Lighting scene is a process of light simulation and how the light emits to the objects in the scene. It is crucial to provide a mood and making a visually appealing output. We have different lighting scenes:

1. **No Lighting** - There are no lights in the scene and there is no interaction between the light and the object in the scene.

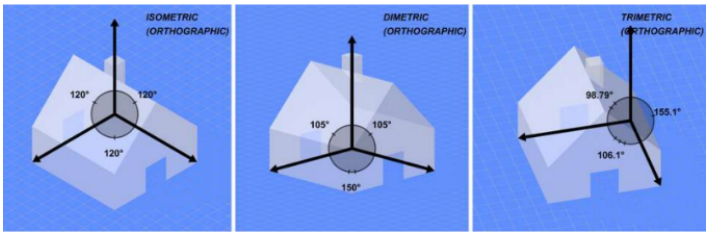
2. **Ambient Light** - It is a non-directional light sources and the light reflects many times from many surfaces which it appears equally from all directions.

CAMERA MODELS

- Insert Camera models are mathematical descriptions of how a 3D scene is projected onto a 2D image. They allow us to control the viewpoint, perspective, and field of view, just like a real camera which are essential for rendering virtual environments.

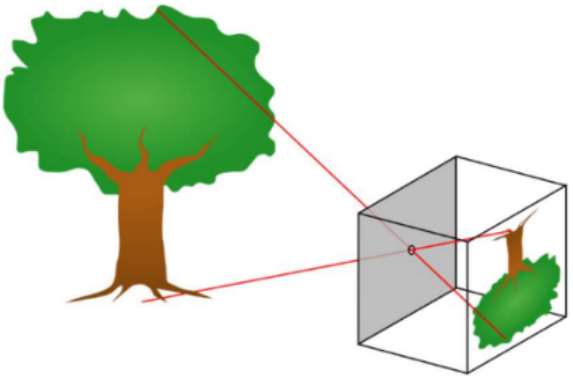
TYPES OF CAMERA MODELS

1. **Orthographic Projection** - all the projection lines are parallel, and there's no perspective foreshortening.



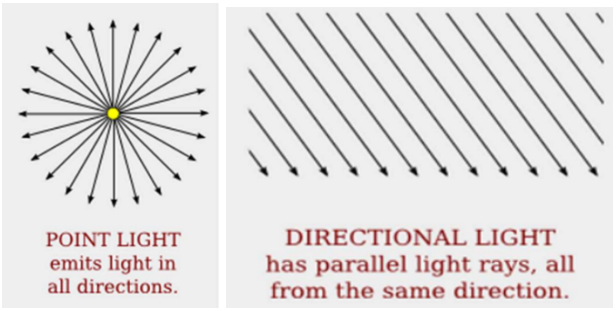
2. **Perspective Projection -** The most common type of camera model and it's designed to mimic how our eyes see the world (Objects appear smaller as they get farther away, and parallel lines converge at a vanishing point).

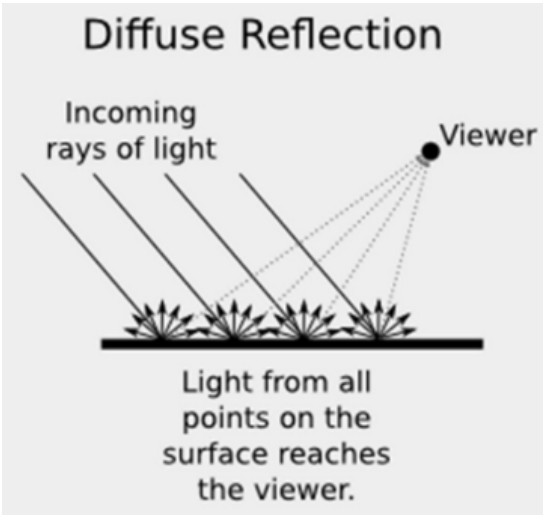
- In the perspective projection, we have a Pinhole Camera Model, the simplest perspective camera model which uses similar geometry to project 3D points onto a 2D image plane.



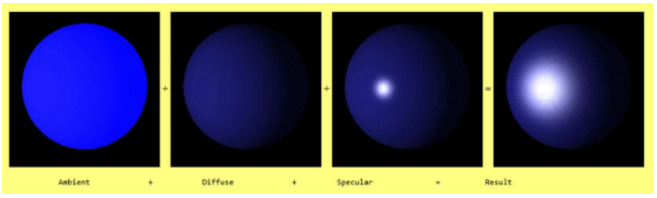
SIMPLE REFLECTION MODELS

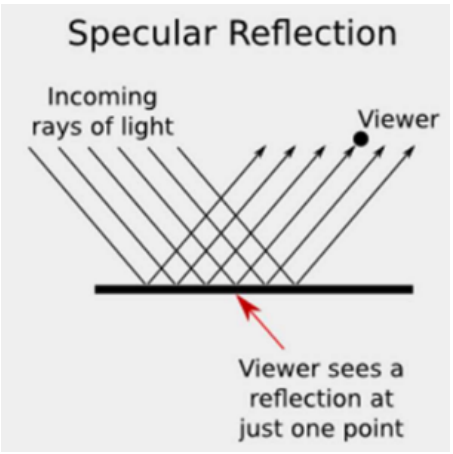
1. **Diffuse Reflection (Lambertian Reflection)**
    - When it uses a point light where it comes from a specific direction. It reflects equally in all direction. While, if it uses a directional light, is known faster than the point light, no computation needed for each polygon.





2. **Specular Reflection**
    - It is a reflection off of shiny surfaces such as shiny metal or plastic that is high in specular component. The position of the viewer is very important for the specular reflection.





LIGHTING TYPES

1. Lighting and Shading provides additional mood, realistic, and visually appearing scene we see in the screen. There are two types of lighting: hard lighting and soft lighting.
    a. **Hard Lighting** creates a sharp and well-defined shadows. While **soft lighting** produces more gradual transitions between the light and shadow.

Lighting types are achieved depending to the light sources used to illuminate light behavior from the source to object.

Types of Light Sources

1. **Directional Light** - A light that is produced by a light source where the light rays emanating from a single parallel direction.
2. **Point Light** - Provides an equal amount of light in all directions. Example of this was the bulb light.
3. **Spotlight** - It radiates the light in a cone shape which is like a flash light.

SHADING MODELS

- This determines the color of a 3D objects surface by calculating the lighting, material properties, etc. which creates a realistic scene. Crucial to provide a realistic and appealing visualization.

Types of Shading Models

1. **Flat Shading (Constant Shading)**
   - It calculates the color once of each face of 3D polygon. Very fast but unrealistic.
2. **Gouraud Shading (Interpolated Shading)**
   - It calculates the color in each vertex then interpolate these colors across the face.
   - Provide smoother appearance of shading.
3. **Phong Shading (Normal Interpolation Shading)**
   - It interpolates the surface normal across the polygon and calculate the lighting equation at each pixel. It creates more accurate and realistic shading.

ANIMATION

- Animation is an art of bringing the object to life in sequence of images or creating the illusion of movement. Animation provides visual communication to make things easier to understand.

TYPES OF ANIMATION

1. **Traditional Animation (Cel Animation)**
   - A classic form of animation where each frame is hand-drawn in transparent celluloid sheets (known as cels) that is placed in top of the painted background. This process creates an illusion of movement.
2. **2D Animation**
   - Animation in two-dimensional space using software which mimics the process of traditional animation which is much faster and cost-effective.
3. **3D Animation**
   - Involves creating an animation in three-dimensional models or environment within a software. This is widely used for films, video games, and television.
4. **Stop Motion Animation**
   - It involves physically manipulating the real-world objects by small increments of photographed frames which appear to move in their own. These includes the following techniques:
     a. **Clay Animation (Claymation) -** It uses clay figures.
     b. **Cut-Out Animation** - It uses flat materials like paper or fabric.
     c. **Puppet Animation** - It uses puppets or dolls.

5. Motion Graphics

- This type focuses in creating animated text or graphics elements that is commonly used for commercials or informative vides that emphasizes the visual communication than providing character movement.

CATEGORIES OF ANIMATION

There are different categories for animation:

1. **Narrative Animation**
   - Involves animation that tells a story.
2. **Commercial Animation**
   - Animation that used for advertising and marketing purposes.
3. **Educational Animation**
   - Designed to tech or explain concepts.
4. **Experimental Animation**
   - Animation that explores the abstract ideas and unconventional techniques.
5. **Animated Documentaries**
   - Used for non-fiction stories to protect the anonymity of subject and visualize events where no footage exists.

ANIMATION TECHNIQUES

1. **Traditional Animation (Cel Animation)** - The oldest and most classic animation technique of hand-drawn in transparent

celluloid sheet and place over a painted background.
2. **2D Animation** - Similar with the process of the traditional animation but with the use of software that make a cost-effective and efficient method of animation.
3. **3D Animation** - A process of animation by creating a movement based from the frames and rigging (digital skeleton attached in 3D model that allows manipulation of the model)
4. **Stop Motion** - Involves in physical manipulation of real-world objects by small increments of photographed frames.
5. **Motion Graphics** - Creating animated and graphic elements used for commercials and informative videos.

APPLICATION OF ANIMATION IN COMPUTER GRAPHICS

1. ***Video games*** - Essential for creating immersive and engaging gameplay experiences.
2. ***Movies*** - For creating animated characters and visual elements.
3. ***Advertising*** - Create an engaging and memorable advertising campaigns.
4. ***Education*** - Develops an interactive and engaging learning materials.
5. ***Training*** - Improves the learning process in fields like medicine, aviation, and military.
6. ***Architectural Visualization*** - Provide an architectural animation and make the building design concepts look real.
7. ***Virtual Reality*** - Create a realistic user experience with detailed and colorful visual effects.

2D VS 3D ANIMATION

- It is known that 3D space operates in x, y, and z dimension while in 2D space only have the x-axis and y-axis dimension. Therefore, there are different transformation happened in 2D that is different in 3D, wherein every time there is transformation, scaling, and rotation happened in the 3D, it always involves the 3 dimensions. This is not the same in 2D since it only has 2 dimensions, so only two axes are involved in transformation, scaling, and rotation happened in 2D.

❖ **Perspective Projection** - It is a technique used to represent 3D objects in 2D plane, which creates depth and realism.

❖ **Composite Transformation** - It involves transformations into a single transformation. For instance, rotating an object and scaling it produces a different result that scaling it first and rotating it.

DYNAMICS

- It refers to the simulation of motion and forces affecting objects in the scene. This is done by calculating the objects mass, velocity, acceleration, and external forces.
- This method enabled us to create realistic animations and simulations.

RENDERING, PIPELINE AND BASIC APPROACH

- It is also known as graphics pipeline. It is a series of steps that transform a 3D scene into an image. This undergoes several stages:

1. **Application Stage** - 3D scene was set up with model and applying the transformation.
2. **Geometry Stage** - Process the geometry of the scene including the vertex transformation, clipping, and projection.
3. **Rasterization Stage** - Converts the processed geometry into pixels.
4. **Pixel Processing Stage** - Applying textures, shades, and light to each pixel.
5. **Output Stage** - Merges the processed pixels into final image.

REAL-TIME RENDERING

- It is a quick process of generating images for interactive viewing and manipulation. This is essential for video games, virtual reality, and interactive simulations. It is done by efficient algorithms and hardware accelerations to achieve high frame rates.

ILLUMINATION MODELS - This determine how light interacts with surfaces in scene to produce

shadows and color as it makes realistic lighting effects. There are common illumination models such as:

1. **Ambient Lighting** - Providing base level illumination.
2. **Diffuse Lighting** - Simulates light scattered evenly from a surface and dependent on the angle between the light and the surface.
3. **Specular Lighting** - It provide special behavior of reflection of light in shiny surfaces.

RADIOSITY - global illumination technique that calculates the inter-reflection of light between surfaces in a scene. In addition, radiosity considers the indirect lighting caused by light bouncing off surfaces, resulting in more realistic and natural-looking lighting. This is done in a method that is computationally intensive but produces high-quality results with soft shadows and accurate color bleeding.

COSC 75

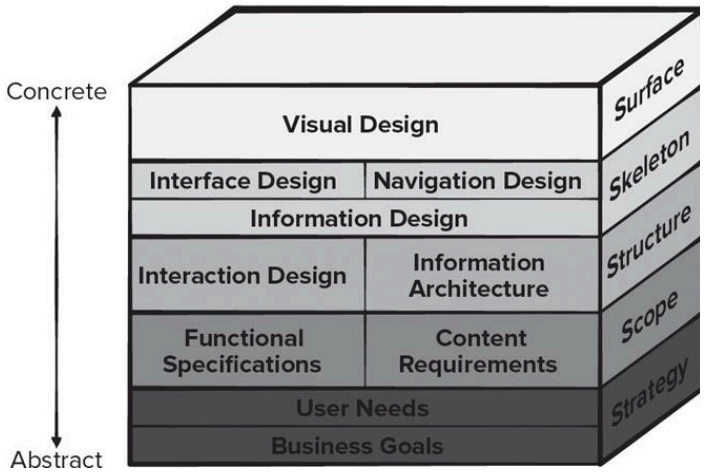
COSC 75

CHAPTER 3: SOFTWARE DESIGN CONCEPTS AND PRINCIPLES

USER EXPERIENCE DESIGN ELEMENTS

- User experience design tries to ensure that no aspect of your software appears the final product without the explicit decision of stakeholders to include it.

- **Strategy.** Identifies user needs and customer business goals that form the basis for all U X design work.
- **Scope.** Includes both the functional and content requirements needed to realize a feature set consistent with the project strategy.
- **Structure.** Consists of the interaction design [For example, how the system reacts in response to user action] and information architecture.
- **Skeleton.** Comprised of three components: information design, interface design, navigation design.
- **Surface.** Presents visual design or the appearance of the finished project to its users.



USER INTERACTION DESIGN

- **Interaction design** focuses on interface between product and user.
- Modes of user input and output include voice input, computer speech generation, touch input, 3D printed output, immersive augmented reality experiences, and sensor tracking of users.
- User interaction should be defined by the stakeholders in the user stories created to describe how users can accomplish their goals using the software product.
- User interaction design should also include a plan for how information should be presented within such a system and how to enable the user to understand that information.
- It is important to recall that the purpose of the user interface is to present just enough information to help the users decide what their next action should be to accomplish their goal and how to perform it.

USABILITY ENGINEERING

- **Usability engineering** is part of U X design work that defines the specification, design, and testing of the human-computer interaction portion of a software product.
- This software engineering action focuses on devising human-computer interfaces that have high usability.
- If developers focus on making a product easy to learn, easy to use, and easy to remember over time, usability can be measured quantitatively and tested for improvements in usability.
- **Accessibility** is the degree to which people with special needs are provided with a means to perceive, understand, navigate, and interact with computer products.
- Accessibility is another aspect of usability engineering that needs to be considered during design.

VISUAL DESIGN

- Visual design (aesthetic design or graphic design) is an artistic endeavor that complements the technical aspects of the user experience design.
- Without it, a software product may be functional, but unappealing.
- With it, a product draws its users into a world that embraces them on an emotional as well as an intellectual level.
- Graphic design considers every aspect of the look and feel of a web or mobile app.

- Not every software engineer has artistic talent. If you fall into this category, hire an experienced graphic designer to help.

Golden Rule 1: Place User in Control

- Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
- Provide for flexible interaction.
- Allow user interaction to be interruptible and undoable.
- Streamline interaction as skill levels advance and allow the interaction to be customized.
- Hide technical internals from the casual user.
- Design for direct interaction with screen objects.

Golden Rule 2: Reduce User's Memory Load

- Reduce demand on short-term memory.
- Establish meaningful defaults.
- Define shortcuts that are intuitive.
- The visual layout of the interface should be based on a real-world metaphor.
- Disclose information in a progressive fashion.

Golden Rule 3: Make Interface Consistent

- Allow the user to put the current task into a meaningful context.
- Maintain consistency across a family of applications.
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

USER INTERFACE DESIGN MODELS

- **User model** — a profile of all end users of the system.
- **Design model** — a design realization of the user model.
- **Mental model (system perception)** — the user's mental image of what the interface is.
- **Implementation model** — the interface "look and feel" coupled with supporting information that describe interface syntax and semantics.

An interface designer needs to reconcile these models and derive a consistent representation of the interface.

USER INTERFACE ANALYSIS AND DESIGN

- ***Interface analysis*** focuses on the profile of the users who will interact with the system.
- ***Interface design*** defines a set of interface objects and actions that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system.
- ***Interface construction*** normally begins with the creation of a prototype that enables usage scenarios to be evaluated.
- ***Interface validation*** focuses on:
    1. The ability of the interface to implement every user task correctly.
    2. The degree to which interface is easy to use and easy to learn.
    3. The user's acceptance of the interface as a tool in her work.

USER EXPERIENCE ANALYSIS

In the case of user experience design, understanding the problem means understanding:

1. the people (end users) who will interact with the system through the interface.
2. the tasks that end users must perform to do their work.
3. the content that is presented as part of the interface.
4. the environment in which these tasks will be conducted.

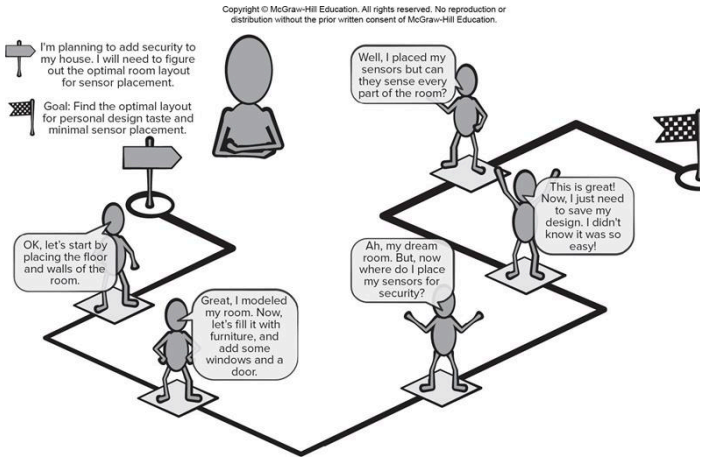
USING CUSTOMER JOURNEY MAP

1. **Gather stakeholders.**
2. **Conduct research.** Collect all information you can about all the things users may experience as they use the software product and define your customer phases (touchpoints).
3. **Build the model.** Create a visualization of the touch points.
4. **Refine the design.** Recruit a designer to make the deliverable visually appealing

and ensure touchpoints are identified clearly.

5. **Identify gaps.** Note any gaps in the customer experience or points of friction or pain (poor transition between phases).
6. **Implement your findings.** Assign responsible parties to bridge the gaps and resolve pain points found.

CUSTOMER JOURNEY MAP



TASK ANALYSIS

Goal of task (scenario) analysis is to answer the following questions:

- What work will the user perform in specific circumstances?
- What tasks and subtasks will be performed as the user does the work?
- What specific problem domain objects will the user manipulate as work is performed?
- What is the sequence of work tasks—the workflow?
- What is the hierarchy of tasks?

USABILITY GUIDELINES

- **Anticipation.** An application should be designed so that it anticipates the user's next move.
- **Communication.** The interface should communicate the status of any activity initiated by the user.
- **Consistency.** The use of navigation controls, menus, icons, and aesthetics (For example, color, shape, layout) should be consistent throughout.
- **Controlled Autonomy.** The interface should facilitate user movement throughout the application, but it should do so in a manner that enforces navigation conventions that have been established for the application.
- **Efficiency.** The design of the application and its interface should optimize the user's work efficiency.
- **Flexibility.** The interface should be flexible enough to enable some users to accomplish tasks directly and others to explore the application in a somewhat random fashion.
- **Focus.** The interface (and the content it presents) should stay focused on the user task(s) at hand.
- **Human Interface Objects.** A vast library of reusable human interface objects has been developed for both Web and mobile apps. Use them.
- **Latency Reduction.** Rather than making the user wait for some internal operation to complete (for example, downloading a complex graphical image), the application should use multitasking in a way that lets the user proceed with work as if the operation has been completed.
- **Learnability.** An application interface should be designed to minimize learning time and, once learned, to minimize relearning required when the app is revisited.
- **Metaphors.** An interface that uses an interaction metaphor is easier to learn and easier to use, as long as the metaphor is appropriate for the application and the user.
- **Readability.** All information presented through the interface should be readable by young and old.
- **Track State.** When appropriate, the state of the user interaction should be tracked and stored so that a user can log off and return later to pick up where he left off.
- **Visible Navigation.** A well-designed interface provides the illusion that users are in the same place, with the work brought to them.

ACCESSIBILITY GUIDELINES

- **Application Accessibility.** Software engineers must ensure that interface design encompasses mechanisms that enable easy for people with special needs.
- **Response Time.** System response time has two important characteristics: length

and variability. Aim for consistency to avoid user frustration.
- **Help Facilities.** Modern software should provide online help facilities that enable a user to get a question answered or resolve a problem without leaving the interface.

- **Error Handling.** Every error message or warning produced by an interactive system should: use user understandable jargon, provide constructive error recovery advice, identify negative consequences of errors, contain an audible or visual cue, and never blame user for causing the error.
- **Menu and Command Labeling.** The use of window-oriented, point-and-pick interfaces has reduced reliance on typed commands. How every it is important to: ensure every menu option has a command version, make commands easy for users to type, make commands easy to remember, allow for command abbreviation, make sure menu labels are self-explanatory, make sure submenus match style of master menu items, and ensure command conventions work across the family of applications.
- **Internationalization.** Software engineers and their managers invariably underestimate the effort and skills required to create user interfaces that accommodate the needs of different locales and languages.

CHAPTER 4: SOFTWARE TESTING STRATEGIES

- Software testing uses different levels, strategies, and techniques to ensure software quality.
    - **Testing Techniques** -how test cases are designed (black-box, white-box)
    - **Testing Levels** - what we test (unit, integration, system, acceptance)
    - **Testing Strategies** - how we integrate and test components (top-down, bottom-up, CI, smoke, regression)

TESTING FUNDAMENTALS

- Attributes of a good test:
- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be “best of breed.”
- A good test should be neither too simple nor too complex.

Two Ways to Test an Engineered Product

Engineered products—machines, devices, or software systems—can be tested from two perspectives:

1. **Based on what the product is supposed to do.**
    - Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function.
2. **Based on how the product works internally.**
    - Knowing the internal workings of a product, tests can be conducted to ensure that “all gears mesh,” that is, internal operations are performed according to specifications and all internal components have been adequately exercised.

CATEGORIES OF TESTING

1. Testing Based on the Specified Function (Black-Box Testing)

- This method assumes you only know what the product should do, not how it works inside.
- Concept: You check whether the product correctly performs each function written in its requirements or specifications.
- Goal: Confirm that every function works as expected. Find errors in outputs, behavior, or performance.

Example

If you are testing a calculator app:

1. Does it add numbers correctly?

2. Does it handle decimal input?
3. Does it show an error for invalid input?

You don't look at the code—you only test inputs and outputs.

2. Testing Based on Internal Workings (White-Box Testing)

- This method assumes you know exactly how the product functions internally.
- White-box testing, is an integration testing philosophy that uses implementation knowledge of the control structures described as part ofcomponent-leveldesignto derive test cases.
- Concept: You examine how the internal components, parts, or code work together.
- Goal: Ensure all internal paths, conditions, loops, calculations, and interactions operate correctly. Verify that all components have been exercised and no part is untested.

Example

If you are testing the calculator's code:

1. Check every code branch (if/else)
2. Check loops and data processing
3. Verify internal algorithms

INTEGRATION TESTING

Integration testing verifies that unit-tested components work together correctly. The objective is to take unit-tested components and build a program structure that matches the design.

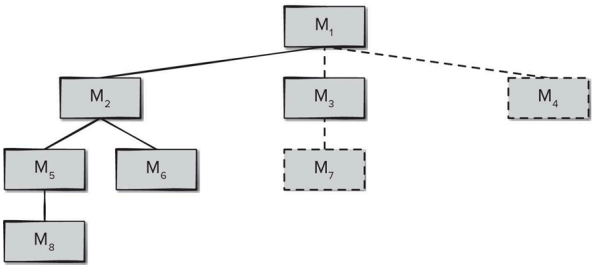
Why It Matters? Most errors occur at interactions:

- Wrong data passed between modules
- Incorrect function calls
- Timing issues
- Interface mismatches

5 INTEGRATION TESTING STRATEGIES

1. Top-Down Integration

- Top-down integration testing is an incremental approach to construction of the software architecture.
- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).
- Modules subordinate to the main control module are incorporated into the structure followed by their subordinates.



How it Works?

- Start from the main control module
- Move downward through the hierarchy
- Use stubs to simulate missing lower modules
- Replace stubs with real modules one at a time

Variants

- Depth-First: Follow one branch deeply before others
- Breadth-First: Complete each level before moving down

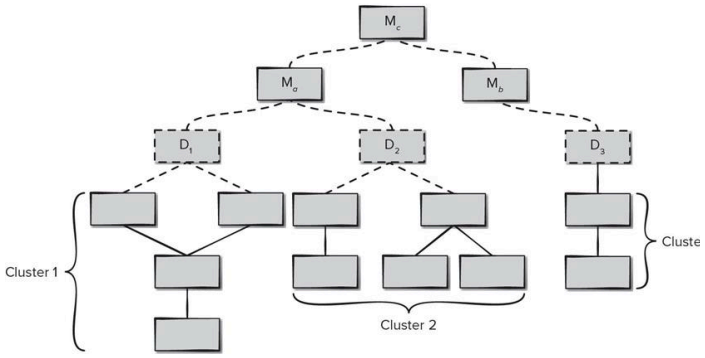
1. The main control module is used as a test driver, and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (for example, depth or breadth first), subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing may be conducted to ensure that new errors have not been introduced.

2. Bottom-Up Integration Testing

- Bottom-up integration testing, begins construction and testing with atomic modules components at the lowest levels in the program structure.

1. Low-level components are combined into clusters (builds) that perform a specific software subfunction.

2. A driver (a control program for testing) is written to coordinate test-case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined, moving upward in the program structure.



How it Works?

- Start testing low-level modules first
- Combine into small functional clusters (builds)
- Use drivers to simulate higher modules
- Move upward until system is complete

3. Continuous Integration

- A practice where developers merge code frequently, usually daily.

Testing Role:

- Each integration triggers automated builds and tests
- Helps detect integration errors early
- Supports Agile, XP, DevOps

4. Smoke Testing Integration

- Software components that have been translated into code are integrated into a build. – that includes all data files, libraries, reusable modules, and components required to implement one or more product functions.
- **Purpose:** A quick initial test of a new build to check basic system stability.
- **Goal:** Verify the build doesn't "crash immediately". Catch showstopper errors early. Ensure the system is testable
- **Benefits:** Reduces integration risk. Improves quality. Simplifies error diagnosis. Shows daily progress.

5. Regression Testing

- Re-running previously executed tests to ensure new changes do NOT break existing functionality.
- Why Necessary? Fixing a bug may affect another area. Adding a feature may create side effects.
- Methods: Manual re-execution. Automated capture/playback. AI-based test selection.

Integration Testing Work Products

- Test Specification
  - Contains test plan + procedures
  - Defines test environment, test cases, and expected results
- Incremental Build Plan
  - Steps for assembling the system
  - Defines order of integration
- Test Report
  - Logs results, issues, resolutions
  - Often shared online for team access

Object-Oriented (OO) Integration Testing

- OO systems require special integration strategies.
- **Thread-Based Testing**
  - Integrates classes needed for one system event or input
  - Tests one thread at a time
  - Regression testing prevents side effects
  - **Example System:** Online Shopping Application
- User event (thread):
- "Place an Order"
  - To process this event, the system uses these classes:
  - Cart
  - Inventory
  - Payment Processor
  - Order Manager
  - Notification Service

How Thread-Based Testing Works:

1. Identify the thread from the use case:Place Order
2. Find all classes required in this thread:

- Cart → gathers items
- Inventory → checks stock
- PaymentProcessor → charges customer
- OrderManager → creates order
- NotificationService → sends email

3. Integrate these classes together (only them).
- Test the entire flow as one combined scenario:
- Add items to cart
- Check inventory
- Process payment
- Create order
- Send confirmation
- When a new thread is added (e.g., “Cancel Order”), run regression testing to ensure the first thread still works.
- Thread-based = testing one complete user interaction at a time.

- **Use-Based Testing**
  - Start with independent classes (few dependencies)
  - Test dependent classes next
  - Build system layer by layer

Example
- Start by testing independent classes → ones that do not rely on others.Then test dependent classes → ones that rely on these independent classes.Build the system layer by layer.
- Use-based = test classes in dependency order: independent → dependent → full system.

Example System: Library Management System

Step 1 — Identify Independent Classes
- These can work alone or use very few other classes:
- Book, Member, LibraryCard

These classes mostly store data and contain simple logic.

Step 2 — Test Independent Classes First

Test them individually:
- Book: title, author, ISBN, availability
- Member: name, ID
- LibraryCard: card number,

Step 3 — Identify Dependent Classes
- These rely on the independent classes:
- LoanManager (depends on Book + Member)
- ReservationManager (depends on Book + Member)
- FineCalculator (depends on Member + LoanManager)

Step 4 — Test Dependent Classes After Independent Ones

Examples:

Test LoanManager
- borrowBook(member, book)
- returnBook(member, book)

Test ReservationManager
- reserveBook(member, book)

Test FineCalculator
- calculateFine(member, overdueDay

Step 5 — Build Up Layers
- Independent layer tested first
- Dependent layer tested next
- Combine layers into full system

Strategy	What It Tests	Example Idea
Thread-Based	One complete user event	“Place an order” thread
Use-Based	Class dependency layers	Independent classes → dependent classes

OO Testing – Fault-Based Test Case Design
- The object of fault-based testing is to design tests that have a high likelihood of uncoveringplausiblefaults.
- Because the product or system must conform to customer requirements, fault-based testing begins with the analysis model.
- The strategy for fault-based testing is to hypothesize a set of plausible faults and then derive tests to prove each hypothesis.
- To determine whether these faults exist, test cases are designed to exercise the design or code.
- **Focus:** Incorrect operations, Wrong message passing, Unexpected results
- **Testing ensures:** Client code calls the correct server operations, Message connections are correct

Scenario-Based Testing
- Testing driven by use cases and user interactions.

- Purpose: Uncover interaction errors, Simulate real-world use, Often exercises multiple subsystems

Example:
Banking app → login → check balance → withdraw → logout

Validation Testing

- validation ensures the final integrated system meets user-visible requirements. Activities:
- Test based on user stories + acceptance criteria
- Generate a deficiency list for deviations
- Conduct a configuration audit to confirm correct final build

Category	Items
Testing Techniques	Black-box, White-box
Testing Levels	Unit, Integration, System, Acceptance
Integration Strategies	Top-Down, Bottom-Up, Big Bang, CI, Smoke, Regression
OO Integration Techniques	Thread-Based, Use-Based, Fault-Based, Scenario-Based
Validation	Final user-level confirmation

Example Activity
You are developing a simple banking application with these features:

- Create account
- Deposit money
- Withdraw money
- Check balance

Rules:

- Cannot withdraw more than current balance
- Deposits and withdrawals must be positive
- Account creation requires a valid username and initial deposit

Step 1: Test Case Design

Module	Test Case Description	Type (Black/White-box)	Expected Outcome / Error Handling
Account Creation	Create account with valid username & deposit	Black-box	Account created successfully
Account Creation	Create account with empty username	Black-box	Error: "Username required"
Deposit	Deposit 500 into account	Black-box	Balance increases by 500
Deposit	Deposit -100 into account	Black-box	Error: "Invalid amount"
Withdrawal	Withdraw 200 when balance is 500	Black-box	Balance decreases by 200
Withdrawal	Withdraw 600 when balance is 500	Black-box	Error: "Insufficient funds"
Check Balance	Check balance after transactions	White-box	Correct balance displayed

Step 2: Testing Levels

Module	Unit	Integration	System	Acceptance
Account Creation	[x]	[ ]	[ ]	[x]
Deposit	[x]	[x]	[ ]	[x]
Withdrawal	[x]	[x]	[ ]	[x]
Check Balance	[x]	[x]	[x]	[x]

Step 3: Integration Strategy

- **Chosen Strategy:** Top-Down
  Reason: Start with the main workflow (Account → Deposit → Withdraw → Balance) and then integrate individual modules to test in sequence.

Step 4: Object-Oriented Integration Techniques

Module	Thread-Based	Use-Based	Fault-Based	Scenario-Based
Account Creation	[ ]	[x]	[x]	[x]
Deposit	[ ]	[x]	[x]	[x]
Withdrawal	[ ]	[x]	[x]	[x]
Check Balance	[ ]	[x]	[ ]	[x]

Step 5: Debugging & Error Handling

Bug ID	Module	Description	Severity	Resolution / Fix
001	Withdrawal	Negative balance allowed	High	Added validation to prevent overdraft
002	Deposit	Negative deposit accepted	Medium	Added error message: "Invalid amount"
003	Account	Empty username allowed	High	Added check: username required

Step 6: Validation / User Testing

Steps a user would perform:

1. Create a new account with valid username and initial deposit 1000
2. Deposit 500 and withdraw 200
3. Check balance (expected 1300)

Result / Confirmation:

- User confirms deposits and withdrawals update balance correctly
- Errors correctly displayed for invalid inputs
- Application workflow works as expected

CHAPTER 5: MAINTENANCE AND REENGINEERING

SOFTWARE MAINTENANCE

- Software is released to end-users, and
  - within days, bug reports filter backto the software engineering organization.
  - within weeks, one class of users indicates that the software must be changed so that it can accommodate the special needsof their environment.
  - within months, another corporate group who wanted nothing to do with the software when it was released, now recognizes that it may provide them with unexpected benefit. They'll need a few enhancements to make it work in their world.
- All of this work is software maintenance

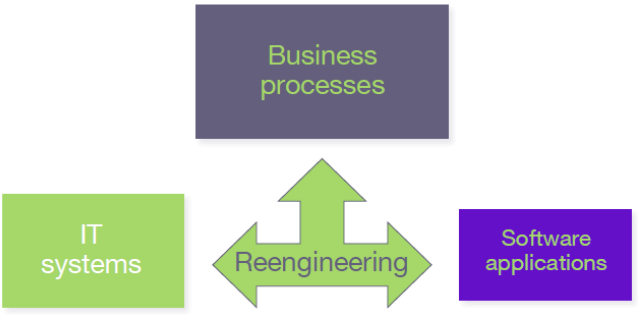
MAINTAINABLE SOFTWARE

- Maintainable software exhibits effective modularity.
- It makes use of design patterns that allow ease of understanding.
- It has been constructed using well-defined coding standards and conventions, leading to source code that is self-documenting and understandable.
- It has undergone a variety of quality assurance techniques that have uncovered potential maintenance problems before the software is released.
- It has been created by software engineers who recognize that they may not be around when changes must be made.
- Therefore, the design and implementation of the software must "assist" the person who is making the change

SOFTWARE SUPPORTABILITY

- "the capability of supporting a software system over its whole product life.
- This implies satisfying any necessary needs or requirements, but also the provision of equipment, support infrastructure, additional software, facilities, manpower, or any other resource required to maintain the software operational and capable of satisfying its function." [SSO08]
- The software should contain facilities to assist support personnel when a defect is encountered in the operational environment (and make no mistake, defects will be encountered).
- Support personnel should have access to a database that contains records of all defects that have already been encountered—their characteristics, cause, and cure.

REENGINEERING

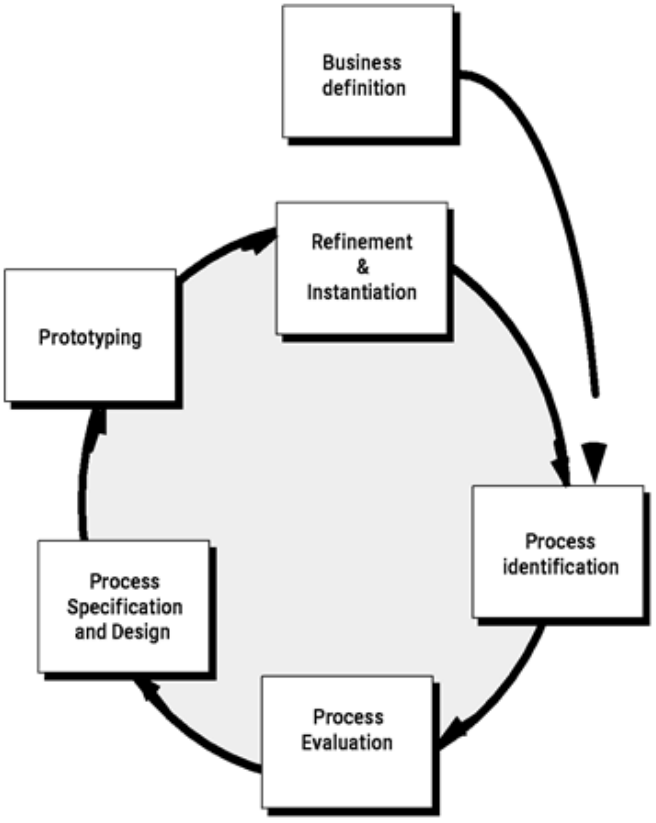


BUSINESS PROCESS REENGINEERING

- **Business definition.** Business goals are identified within the context of four key drivers: cost reduction, time reduction, quality improvement, and personnel development and empowerment.

- **Process identification.** Processes that are critical to achieving the goals defined in the business definition are identified.
- **Process evaluation.** The existing process is thoroughly analyzed and measured.
- **Process specification and design.** Based on information obtained during the first three BPR activities, use-cases are prepared for each process that is to be redesigned.
- **Prototyping.** A redesigned business process must be prototyped before it is fully integrated into the business.
- **Refinement and instantiation.** Based on feedback from the prototype, the business process is refined and then instantiated within a business system.

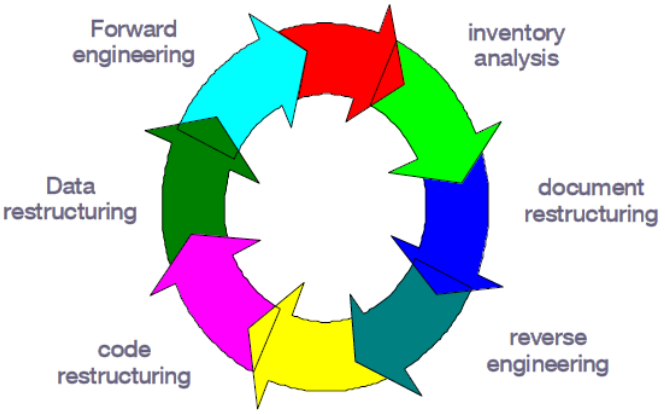
BUSINESS PROCESS REENGINEERING



BPR PRINCIPLES

- Organize around outcomes, not tasks.
- Have those who use the output of the process perform the process.
- Incorporate information processing work into the real work that produces the raw information.
- Treat geographically dispersed resources as though they were centralized.
- Link parallel activities instead of integrated their results. When different
- Put the decision point where the work is performed, and build control into the process.
- Capture data once, at its source.

SOFTWARE REENGINEERING



INVENTORY ANALYSIS

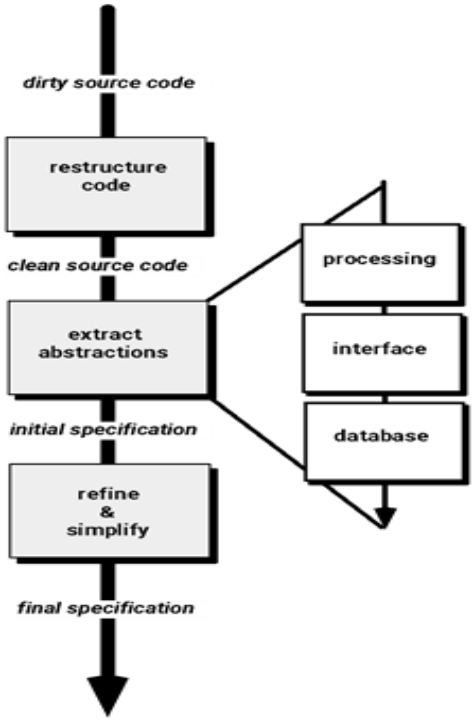
- build a table that contains all applications
- establish a list of criteria, e.g.,
    - name of the application
    - year it was originally created
    - number of substantive changes made to it
    - total effort applied to make these changes
    - date of last substantive change
    - effort applied to make the last change
    - system(s) in which it resides
    - applications to which it interfaces, …
- analyze and prioritize to select candidates for reengineering

DOCUMENT RESTRUCTURING

- Weak documentation is the trademark of many legacy systems. But what do we do about it? What are our options? Options …
- Creating documentation is far too time consuming.If the system works, we'll live with what we have. In some cases, this is the correct approach.

- Documentation must be updated, but we have limited resources. We'll use a "document when touched" approach. It may not be necessary to fully redocument an application.
- The system is business critical and must be fully redocumented. Even in this case, an intelligent approach is to pare documentation to an essential minimum.

REVERSE ENGINEERING



CODE RESTRUCTURING

- Source code is analyzed using a restructuring tool.
- Poorly design code segments are redesigned
- Violations of structured programming constructs are noted and code is then restructured (this can be done automatically)
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced
- Internal code documentation is updated.

DATA RESTRUCTURING

- Unlike code restructuring, which occurs at a relatively low level of abstraction, data structuring is a full-scale reengineering activity
- In most cases, data restructuring begins with a reverse engineering activity.
    - Current data architecture is dissected and necessary data models are defined (Chapter 9).
    - Data objects and attributes are identified, and existing data structures are reviewed for quality.
    - When data structure is weak (e.g., flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered.

- Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.

FORWARD ENGINEERING

1. The cost to maintain one line of source code may be 20 to 40 times the cost of initial development of that line.
2. Redesign of the software architecture (program and/or data structure), using modern design concepts, can greatly facilitate future maintenance.
3. Because a prototype of the software already exists, development productivity should be much higher than average.
4. The user now has experience with the software. Therefore, new requirements and the direction of change can be ascertained with greater ease.
5. CASE tools for reengineering will automate some parts of the job.
6. A complete software configuration (documents, programs and data) will exist upon completion of preventive maintenance.

COSC 80

Lecture 6 - Shortest Remaining Time First

SHORT REVIEW:

- **First Come First Serve** – whichever the process enters the ready queue first is executed first. Follows the First In First Out principle.
- **Shortest Job First** – is the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

SHORTEST REMAINING JOB FIRST

- The preemptive version of SJF scheduling, where it will preempt the currently executing process.
- In SRTF, the execution of the process can be **stopped** after a certain amount of time. At the arrival of every process, the **short term scheduler schedules** the process with the least remaining burst time among the list of available processes and the running process.
- Once all the process are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF Scheduling.**
- The context of the process is saved in the **PCB (Process Control Block)** when the process is removed from the execution and the next process is scheduled.

ADVANTAGES:

- SRTF Algorithm makes the processing of the jobs faster than SJF algorithm, given it's overhead charges are not counted.
    - Scheduling Overhead means the time taken by dispatcher to move the process from ready state to running state.
- Allows for easier management of library updates or replacements without recompiling the program.
- Enables efficient memory usage, as libraries can be shared among multiple instances of the program.
- Provides better portability, as the program can be executed on different systems with compatible libraries available at runtime.

DISADVANTAGES

- The context switch is done a lot more times in SRTF than in SJF, and consumes CPU's valuable time for processing. This adds up to its processing time and diminishes its advantage of fast processing.
- Slightly slower program startup due to the additional linking process.
- Requires proper handling of library dependencies to ensure correct execution.
- Debugging can be slightly more complex, as libraries are separate entities loaded at runtime.

Lecture 7 - Priority Scheduling

SHORT REVIEW:

- **Shortest Remaining Job First** – The process with the smallest amount of time remaining until completion is selected to execute.
- **Round Robin** – Each process is assigned a fixed time slot in CPU (Time Quantum/Time Slice) in a cyclic way. Basically, a preemptive version of FCFS Scheduling Algorithm.

PRIORITY CPU SCHEDULING

- **Priority Scheduling** is a method of scheduling processes that is based on priority.
    - A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
    - Equal-priority processes are scheduled in FCFS order.

- Priorities are generally indicated by some fixed range of numbers. However, there is no general agreement on whether 0 is the highest or lowest priority.
    - Some systems use low numbers to represent low priority; others use low numbers for high priority.

PREEMPTIVE SCHEDULING

- The tasks are mostly assigned with their priorities.
- It is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running.
- The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

NON-PREEMPTIVE SCHEDULING

- In Non-Preemptive Scheduling, the CPU has been allocated to a specific process.
- The process that keeps the CPU busy, will release the CPU either by switching context or terminating.

CHARACTERISTICS OF PRIORITY SCHEDULING

- In Priority Scheduling, it used in Operating Systems for performing batch processes;
- If two jobs having the same priority are READY, it works on a FCFS basis;
- In priority scheduling, a number is assigned to each process that indicates its priority level;
- In Preemptive priority scheduling, if a newer process arrives, that is having a higher priority than the currently running process, then the currently running process is preempted.

PROBLEM WITH PRIORITY SCHEDULING

- Major problem with priority scheduling is indefinite blocking or starvation.
    - A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low-priority processes waiting indefinitely.
    - In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.
- A solution to the problem of indefinite blockage of low-priority processes is Aging.
    - Aging gradually increasing the priority of processes that wait in the system for a long time.
    - For instance, if priorities range from a certain priority n to n+1, periodically (assuming every second) increase the priority of a waiting process by 1.
- Another option is to combine round-robin and priority scheduling in such a way that the system executes the higher-priority process and runs processes with the same priority using round-robing scheduling.

Lecture 8 - Round Robin Scheduling

ROUND ROBIN

- **Round Robin (RR)** is similar to FCFS scheduling, but **preemption is added** to enable the system to switch between processes.
- A small unit of time, called a **time quantum** or **time slice**, is defined. A time quantum is generally from 10 to 100ms in length.
- The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process of a time interval of up to 1 time quantum.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- In RR, the FIFO (First In First Out) approach is treated in the ready queue.

ADVANTAGE

- It doesn't face the issues of starvation or convoy effect.
- All the jobs get a fair allocation of CPU.
- It deals with all process without any priority.
- This scheduling method does not depend upon burst time. That's why it is easily implementable on the system.
- Once a process is executed for a specific set of the period, the process is preempted, and another process executes for that given time period.
- This Scheduling Algorithm allows OS to use the Context Switching method to saved states of preempted processes.
- It gives the best performance in terms of average response time.

DISADVANTAGE

- If slicing time of OS is low, the processor output will be reduced.
- This method spends more time on context switching.
- Its performance heavily depends on time quantum.
- Priorities cannot be set for the processes.
- Round-Robin Scheduling doesn't give special priority to more important tasks.
- Decreases comprehension
- Lower time quantum results in higher the context switching overhead in the system.
- Finding a correct time quantum is a quite difficult task in this system.

DCIT 65

Alesya